

Session 3

Unsupervised filters and Apriori

Apriori needs nominal attributes, so discretisation comes first. This session runs Apriori across several datasets with different support and confidence settings and studies the rules produced.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 7 to 8 of the manual: unsupervised filters and apriori
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q7	Perform the following	Complete
Q8	Implement the Apriori Algorithm to find the association rules in contactlenses.arff...	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- Apriori parameters in WEKA: `lowerBoundMinSupport`, `upperBoundMinSupport`, `delta`, `minMetric` (confidence), `numRules`. Know what each does before changing it.
- Discretize with equal-width and equal-frequency bins and compare the rules; note that bin boundaries appear in the rules.

- For question 8, implement Apriori by hand or in Python on the 24-row contact-lenses data: candidate generation, support counting, pruning, then rule generation with confidence.

Question 7

Problem Statement

■ Write in lab record

Perform the following:

- Explore various options available in WEKA for preprocessing data and apply unsupervised filters like Discretization, Resample-filter etc. on various datasets.
- Load weather, nominal, Iris, Glass datasets into WEKA and run Apriori algorithms with different support and confidence values.
- Study the rules generated.
- Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated.
- Derive interesting insights and observe the effect of discretization in the rule generation process.

Solution

■ Write in lab record

Steps

Unsupervised filters live under Filter, Choose, `weka.filters.unsupervised`. The ones used in this session:

Filter	Menu path	Options that matter
Discretize	attribute, Discretize	<code>attributeIndices</code> (default first-last, numeric only), <code>bins</code> (10), <code>useEqualFrequency</code> (False), <code>findNumBins</code> (False), <code>makeBinary</code> (False)
Resample	instance, Resample	<code>sampleSizePercent</code> (100), <code>noReplacement</code> (False), <code>randomSeed</code> (1)
Normalize	attribute, Normalize	<code>scale</code> 1.0, <code>translation</code> 0.0
Standardize	attribute, Standardize	none; gives mean 0, standard deviation 1
NominalToBinary	attribute, NominalToBinary	<code>transformAllValues</code> , <code>attributeIndices</code>
RemoveUseless	attribute, RemoveUseless	<code>maximumVariancePercentageAllowed</code> 99; drops constant attributes and identifiers
Randomize	instance, Randomize	<code>randomSeed</code> ; shuffles row order

Apriori is at Associate tab, Choose, `weka.associations.Apriori`. Click the name to open the options:

Parameter	Default	Meaning
upperBoundMinSupport	1.0	support at which the search starts
delta	0.05	support is lowered by this each cycle
lowerBoundMinSupport	0.1	search stops here
metricType	Confidence	Confidence, Lift, Leverage or Conviction
minMetric	0.9	rules below this are dropped
numRules	10	stop lowering support once this many rules pass
outputItemSets	False	also print the large itemsets
car	False	mine class association rules only

1. weather.nominal, run 1: Open file `weather.nominal.arff`, Associate, Choose Apriori, keep defaults, Start.
2. weather.nominal, run 2: click the Apriori name, set `lowerBoundMinSupport` 0.3 and `minMetric` 0.7, OK, Start.
3. iris: Open file `iris.arff`, Preprocess, Discretize with `bins` 3 on first-last (the class is nominal and is skipped), Apply. Associate, Apriori defaults, Start. Then `minMetric` 0.7, `numRules` 8, Start.
4. iris again with the default 10 bins: Undo, Discretize with `bins` 10, Apply, Apriori defaults, Start. Compare with step 3.
5. glass: Open file `glass.arff`, Discretize `bins` 3, Apply, Apriori defaults, Start; then `lowerBoundMinSupport` 0.5, `minMetric` 0.95, Start.
6. Every output below that says computed was produced with `apriori.py` (Question 8) on the same data; the Python and WEKA output formats are the same, so compare line by line.

Output

weather.nominal, defaults (computed; identical to the block on page 57 of the manual):

```

1 Minimum support: 0.15 (2 instances)
2 Minimum metric <confidence>: 0.9
3 Number of cycles performed: 17
4
5 Generated sets of large itemsets:
6
7 Size of set of large itemsets L(1): 12
8
9 Size of set of large itemsets L(2): 47
10
11 Size of set of large itemsets L(3): 39
12
13 Size of set of large itemsets L(4): 6
14
15 Best rules found:
16
17 1. outlook=overcast 4  $\implies$  play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1
18 2. temperature=cool 4  $\implies$  humidity=normal 4    <conf:(1)> lift:(2) lev:(0.1
19 3. humidity=normal windy=FALSE 4  $\implies$  play=yes 4    <conf:(1)> lift:(1.56) l
20 4. outlook=sunny play=no 3  $\implies$  humidity=high 3    <conf:(1)> lift:(2) lev:(
21 5. outlook=sunny humidity=high 3  $\implies$  play=no 3    <conf:(1)> lift:(2.8) lev
22 6. outlook=rainy play=yes 3  $\implies$  windy=FALSE 3    <conf:(1)> lift:(1.75) lev
23 7. outlook=rainy windy=FALSE 3  $\implies$  play=yes 3    <conf:(1)> lift:(1.56) lev
24 8. temperature=cool play=yes 3  $\implies$  humidity=normal 3    <conf:(1)> lift:(2)
25 9. outlook=sunny temperature=hot 2  $\implies$  humidity=high 2    <conf:(1)> lift:(
26 10. temperature=hot play=no 2  $\implies$  outlook=sunny 2    <conf:(1)> lift:(2.8) l

```

weather.nominal, `lowerBoundMinSupport` 0.3 and `minMetric` 0.7 (computed):

```

1 Minimum support: 0.3 (4 instances)
2 Minimum metric <confidence>: 0.7
3 Number of cycles performed: 14
4
5 Generated sets of large itemsets:
6
7 Size of set of large itemsets L(1): 12
8
9 Size of set of large itemsets L(2): 9
10
11 Size of set of large itemsets L(3): 1
12
13 Best rules found:
14
15 1. outlook=overcast 4  $\implies$  play=yes 4    <conf:(1)> lift:(1.56) lev:(0.1) [1
16 2. temperature=cool 4  $\implies$  humidity=normal 4    <conf:(1)> lift:(2) lev:(0.1
17 3. humidity=normal windy=FALSE 4  $\implies$  play=yes 4    <conf:(1)> lift:(1.56) l
18 4. humidity=normal 7  $\implies$  play=yes 6    <conf:(0.86)> lift:(1.33) lev:(0.11)
19 5. play=no 5  $\implies$  humidity=high 4    <conf:(0.8)> lift:(1.6) lev:(0.11) [1]
20 6. windy=FALSE 8  $\implies$  play=yes 6    <conf:(0.75)> lift:(1.17) lev:(0.06) [0]

```

Only 6 rules pass: at support 0.3 (4 instances) there are just 9 large 2-itemsets and one 3-itemset, and support cannot drop further because the lower bound was reached.

iris discretised into 3 equal-width bins, Apriori defaults. Computed on `iris-sample.arff`, the first 10 rows of each class (30 of the 150 rows), so the counts are 10 where WEKA prints 50 on the full file:

```

1 Minimum support: 0.3 (9 instances)
2 Minimum metric <confidence>: 0.9
3 Number of cycles performed: 14
4
5 Generated sets of large itemsets:
6
7 Size of set of large itemsets L(1): 12
8
9 Size of set of large itemsets L(2): 12
10
11 Size of set of large itemsets L(3): 6
12
13 Size of set of large itemsets L(4): 1
14
15 Best rules found:
16
17 1. petallength='(-inf-3.066667]' 10 ==> sepallength='(-inf-5.466667]' 10
18 2. petalwidth='(-inf-0.9]' 10 ==> sepallength='(-inf-5.466667]' 10 <conf
19 3. class=Iris-setosa 10 ==> sepallength='(-inf-5.466667]' 10 <conf:(1)>
20 4. petalwidth='(-inf-0.9]' 10 ==> petallength='(-inf-3.066667]' 10 <conf
21 5. petallength='(-inf-3.066667]' 10 ==> petalwidth='(-inf-0.9]' 10 <conf
22 6. class=Iris-setosa 10 ==> petallength='(-inf-3.066667]' 10 <conf:(1)>
23 7. petallength='(-inf-3.066667]' 10 ==> class=Iris-setosa 10 <conf:(1)>
24 8. petallength='(3.066667-4.833333]' 10 ==> petalwidth='(0.9-1.7]' 10 <c
25 9. class=Iris-setosa 10 ==> petalwidth='(-inf-0.9]' 10 <conf:(1)> lift:(
26 10. petalwidth='(-inf-0.9]' 10 ==> class=Iris-setosa 10 <conf:(1)> lift:(

```

With `minMetric` 0.7 and `numRules` 8 the same 8 rules come out first, because there are already more than 8 rules at confidence 1 and lowering the threshold only adds rules further down the list.

On the full 150-row file the bin labels are '(-inf-2.966667]' for petallength and '(-inf-0.9]' for petalwidth, and the top rules are the same pairs with count 50: `petalwidth='(-inf-0.9]' 50 ==> class=Iris-setosa 50 conf:(1)`, `petallength='(-inf-2.966667]' 50 ==> class=Iris-setosa 50 conf:(1)`, and both directions between the two petal bins. Every setosa has petal width at most 0.6, so the lowest bin is exactly the setosa class.

iris sample with the default 10 bins (computed):

```

1 Minimum support: 0.3 (9 instances)
2 Minimum metric <confidence>: 0.9
3 Number of cycles performed: 14
4
5 Generated sets of large itemsets:
6
7 Size of set of large itemsets L(1): 5
8
9 Size of set of large itemsets L(2): 3
10
11 Size of set of large itemsets L(3): 1
12
13 Best rules found:
14
15 1. class=Iris-setosa 10  $\implies$  petallength='(-inf-1.83]' 10 <conf:(1)> lift
16 2. petallength='(-inf-1.83]' 10  $\implies$  class=Iris-setosa 10 <conf:(1)> lift
17 3. petalwidth='(-inf-0.34]' 9  $\implies$  petallength='(-inf-1.83]' 9 <conf:(1)>
18 4. petalwidth='(-inf-0.34]' 9  $\implies$  class=Iris-setosa 9 <conf:(1)> lift:(3
19 5. petalwidth='(-inf-0.34]' class=Iris-setosa 9  $\implies$  petallength='(-inf-1.83
20 6. petallength='(-inf-1.83]' petalwidth='(-inf-0.34]' 9  $\implies$  class=Iris-seto
21 7. petalwidth='(-inf-0.34]' 9  $\implies$  petallength='(-inf-1.83]' class=Iris-seto
22 8. petallength='(-inf-1.83]' 10  $\implies$  petalwidth='(-inf-0.34]' 9 <conf:(0.
23 9. class=Iris-setosa 10  $\implies$  petalwidth='(-inf-0.34]' 9 <conf:(0.9)> lift
24 10. petallength='(-inf-1.83]' class=Iris-setosa 10  $\implies$  petalwidth='(-inf-0.3

```

glass (214 rows, 9 numeric attributes, Discretize bins 3, Apriori defaults; expected shape, not run here): Ba is 0 in 176 rows and Fe is 0 in 144 rows, so the lowest Ba and Fe bins are the most frequent items and the top rules connect them, for example `Fe='(-inf-0.17]'  $\implies$  Ba='(-inf-1.05]'` with confidence near 1 and support above 0.6. With `lowerBoundMinSupport` 0.5 and `minMetric` 0.95 only rules among Ba, Fe and K bins remain, and no rule mentions `Type`, because the largest class (build wind non-float, 76 rows) is only 36 percent of the data.

Explanation

Support and confidence pull in opposite directions. Lowering `minMetric` from 0.9 to 0.7 admits weaker rules such as `windy=FALSE  $\implies$  play=yes` (conf 0.75), whose lift 1.17 says it is barely better than guessing `play=yes` outright (9 of 14). Raising `lowerBoundMinSupport` to 0.3 removes every rule about `temperature=hot` or `outlook=sunny` with `play=no`, because those itemsets cover only 2 or 3 rows.

Discretisation decides what the rules can say. With 3 bins, the iris sample gives 12 large single items and 12 pairs, and the rules read as class descriptions (`petallength` small and `petalwidth` small mean setosa). With 10 bins each bin holds fewer rows, so only 5 items reach the 30 percent support and the rules describe the same setosa cluster with narrower intervals (`(-inf-1.83]` , `(-inf-0.34]`) and lower confidence for the reverse direction (0.9 instead of 1). Finer bins give more precise but rarer items; coarser bins give frequent but vaguer items. Equal-frequency bins (`useEqualFrequency` True) make every item about equally frequent, which removes the tall-bin effect seen on glass.

Interesting rules are the ones with confidence near 1 and lift well above 1: `temperature=cool  $\Rightarrow$  humidity=normal` (lift 2) is a real dependency in the weather data; `outlook=overcast  $\Rightarrow$  play=yes` (lift 1.56) is the rule a decision tree also finds.

Question 8

Problem Statement

■ Write in lab record

Implement the Apriori Algorithm to find the association rules in `contactlenses.arff` dataset.

Solution

■ Write in lab record

Steps

1. Read the ARFF: the header gives the attribute names and the value list, each data row becomes a transaction whose items are the (attribute, value) pairs.
2. Level 1: count every item; keep those with count at least the minimum support count. For support 0.2 on 24 rows the count is $\lceil 0.2 \times 24 + 0.5 \rceil = 5$ (WEKA rounds 4.8 to 5).
3. Level k: join two large (k-1)-itemsets that differ in one item, keep the candidate only if all its (k-1)-subsets are large (the Apriori pruning), then scan the transactions to count it.
4. Stop when a level is empty.
5. Rules: from every large itemset of size 2 or more, try each non-empty proper subset as the consequent; confidence is the itemset count divided by the premise count; keep the rule when confidence is at least 0.9.

6. Rank by confidence, then by support, print the top 10 in WEKA's format.

7. Run: `python3 apriori.py contact-lenses.arff -M 0.2 -U 0.2` fixes the support at 0.2 for one pass. `python3 apriori.py contact-lenses.arff` repeats WEKA's default search from 1.0 downwards.

Program

- Lab record: every tab is one file of the answer. Write all of them.

apriori.py

apriori.py

```

1  #!/usr/bin/env python3
2  """Apriori association-rule miner that behaves like weka.associations.Aprior
3
4  Standard library only. Reads an ARFF file with nominal attributes (numeric
5  attributes are discretised into equal-width bins first, like WEKA's
6  unsupervised Discretize filter) and prints the same blocks WEKA prints.
7
8  Usage:
9      python3 apriori.py FILE.arff [-N rules] [-C minMetric] [-T 0|1] [-D delta]
10                                     [-U upperBoundMinSupport] [-M lowerBoundMinSupport]
11                                     [-B bins] [-R attr,attr] [-I]
12
13      -T 0 ranks by confidence (default), -T 1 by lift. -I prints the large
14      itemsets. -R removes attributes by 1-based index before mining (like the
15      Remove filter). -B is the number of bins for numeric attributes.
16
17  Examples:
18      python3 apriori.py contact-lenses.arff                # WEKA defaults
19      python3 apriori.py contact-lenses.arff -M 0.2 -U 0.2 -I # one pass at 0.2
20      python3 apriori.py iris-sample.arff -B 3 -M 0.3
21  """
22  import argparse
23  import itertools
24  import re
25  from collections import Counter
26  from decimal import ROUND_HALF_UP, Decimal
27
28
29  def read_arff(path):
30      """Return (relation, [(name, values-or-None)], rows). '?' marks missing.
31      relation, attrs, rows, in_data = "", [], [], False
32      with open(path) as f:
33          for line in f:
34              line = line.strip()
35              if not line or line.startswith("%"):
36                  continue
37              if in_data:
38                  rows.append([v.strip().strip("'\"") for v in line.split(",")])
39                  continue
40              low = line.lower()
41              if low.startswith("@relation"):
42                  relation = line.split(None, 1)[1].strip("'\"")
43              elif low.startswith("@attribute"):

```

```

44         m = re.match(r"@attribute\s+(['^']*|\"[^\"]*\"|\S+)\s+(.*)\"
45         name, typ = m.group(1).strip(\"'\").strip(), m.group(2).strip()
46         if typ.startswith("{"):
47             attrs.append((name, [v.strip().strip(\"'\") for v in typ
48             else:
49                 attrs.append((name, None)) # numeric / real / integer
50         elif low.startswith("@data"):
51             in_data = True
52     return relation, attrs, rows
53
54
55 def fmt(x, places=6):
56     """WEKA's Utils.doubleToString: rounds half up, trailing zeros trimmed."
57     s = str(Decimal(repr(x)).quantize(Decimal(1).scaleb(-places), ROUND_HALF
58     return s if s not in ("", "-0") else "0"
59
60
61 def discretize(attrs, rows, bins):
62     """Equal-width bins with WEKA's labels: '(-inf-c1]', '(c1-c2]', '(cn-inf
63     for j, (name, values) in enumerate(attrs):
64         if values is not None:
65             continue
66         nums = [float(r[j]) for r in rows if r[j] != "?"]
67         lo, hi = min(nums), max(nums)
68         width = (hi - lo) / bins
69         cuts = [lo + i * width for i in range(1, bins)]
70         labels = ["'(-inf-%s]" % fmt(cuts[0])]
71         labels += ["'(%s-%s)" % (fmt(a), fmt(b)) for a, b in zip(cuts, cuts
72         labels.append("'(%s-inf)" % fmt(cuts[-1]))
73         for r in rows:
74             if r[j] == "?":
75                 continue
76             v, k = float(r[j]), 0
77             while k < len(cuts) and v > cuts[k]:
78                 k += 1
79             r[j] = labels[k]
80         attrs[j] = (name, labels)
81     return attrs, rows
82
83
84 def large_itemsets(trans, min_count):
85     """Level-wise Apriori. Returns [L1, L2, ...] as dicts itemset → count."
86     counts = Counter(i for t in trans for i in t)
87     levels = [{frozenset([i]): c for i, c in counts.items() if c ≥ min_coun

```

```

88     while levels[-1]:
89         prev = levels[-1]
90         candsets = set()
91         for a, b in itertools.combinations(sorted(prev, key=sorted), 2):
92             u = a | b
93             if len(u) == len(a) + 1 and all(frozenset(s) in prev for s in it
94                 candsets.add(u)
95         # ponytail: O(|L|^2) join, fine for lab-sized files; prefix join if
96         counts = {c: sum(1 for t in trans if c ≤ t) for c in candsets}
97         levels.append({c: n for c, n in counts.items() if n ≥ min_count})
98     return levels[:-1]
99
100
101
102
103 def rules_from(levels, n, min_metric, metric_type):
104
105     """All rules  $X \Rightarrow Y$  from every large itemset, kept when metric  $\geq$  min_m
106
107     Generation order matches WEKA: itemsets in attribute order, consequents
108
109     size 1 first, then size 2, ... (this fixes the tie order in the output).
110
111     supp = {k: v for lev in levels for k, v in lev.items()}
112
113     out = []
114
115     for lev in levels[1:]:
116
117         for iset in sorted(lev, key=sorted):
118
119             items = sorted(iset)
120
121             for size in range(1, len(items)):
122
123                 for cons in itertools.combinations(items, size):
124
125                     cons = frozenset(cons)
126
127                     prem = iset - cons
128
129                     nxy, nx, ny = supp[iset], supp[prem], supp[cons]
130
131                     conf = nxy / nx

```

```

6         lift = conf / (ny / n)
11
7         lev_ = nxy / n - (nx / n) * (ny / n)
11
8         conv = (nx * (n - ny) / n) / (nx - nxy + 1) # WEKA adds
11
9         metric = conf if metric_type == 0 else lift
12
0         if metric >= min_metric - 1e-9:
12
1         out.append((prem, cons, nx, nxy, conf, lift, lev_, c
12
2     return out
12
3
4
12
5 def item_str(attrs, item):
12
6     j, vi = item
12
7     return f"{attrs[j][0]}={attrs[j][1][vi]}"
12
8
9
13
0 def main():
13
1     p = argparse.ArgumentParser()
13
2     p.add_argument("file")
13
3     p.add_argument("-N", type=int, default=10, help="numRules")
13
4     p.add_argument("-C", type=float, default=0.9, help="minMetric")
13
5     p.add_argument("-T", type=int, default=0, help="metricType 0=confidence
13
6     p.add_argument("-D", type=float, default=0.05, help="delta")
13
7     p.add_argument("-U", type=float, default=1.0, help="upperBoundMinSupport

```

```

13
8     p.add_argument("-M", type=float, default=0.1, help="lowerBoundMinSupport
13
9     p.add_argument("-B", type=int, default=10, help="bins for numeric attrib
14
0     p.add_argument("-R", default="", help="1-based attribute indices to remo
14
1     p.add_argument("-I", action="store_true", help="outputItemSets")
14
2     a = p.parse_args()
14
3
14
4     relation, attrs, rows = read_arff(a.file)
14
5     if a.R:
14
6         drop = {int(i) - 1 for i in a.R.split(",")}
14
7         attrs = [x for j, x in enumerate(attrs) if j not in drop]
14
8         rows = [[v for j, v in enumerate(r) if j not in drop] for r in rows]
14
9     attrs, rows = discretize(attrs, rows, a.B)
15
0     # items are (attribute index, value index) so ordering follows the ARFF
15
1     trans = [frozenset((j, attrs[j][1].index(v)) for j, v in enumerate(r) if
15
2     n = len(trans)
15
3
15
4     metric_name = ["confidence", "lift"][a.T]
15
5     print("≡≡≡ Run information ≡≡≡\n")
15
6     print(f"Scheme:          weka.associations.Apriori -N {a.N} -T {a.T} -C {a.
15
7     print(f"Relation:      {relation}")
15
8     print(f"Instances:     {n}")
15
9     print(f"Attributes:     {len(attrs)}")

```

```

16
0     for name, _ in attrs:
16
1         print(f"                {name}")
16
2     print("≡≡≡ Associator model (full training set) ≡≡≡\n\n\nApriori\n=====
16
3
16
4     # WEKA: start at upper - delta, lower the support each cycle until numRu
16
5     # rules pass minMetric or the lower bound is reached.
16
6     min_support, cycles = round(a.U - a.D, 10), 0
16
7     if min_support < a.M:
16
8         min_support = a.M
16
9     while True:
17
10         need = int(min_support * n + 0.5)
17
11         levels = large_itemsets(trans, need)
17
12         rules = rules_from(levels, n, a.C, a.T)
17
13         cycles += 1
17
14         nxt = round(min_support - a.D, 10)
17
15         if len(rules) ≥ a.N or nxt < a.M - 1e-9 or nxt ≤ 0:
17
16             break
17
17         min_support = nxt
17
18
19     print(f"Minimum support: {fmt(min_support, 2)} ({need} instances)")
18
20     print(f"Minimum metric <{metric_name}>: {fmt(a.C, 2)}")
18
21     print(f"Number of cycles performed: {cycles}\n")

```

```

18
2    print("Generated sets of large itemsets:\n")
18
3    for k, lev in enumerate(levels, 1):
18
4        print(f"Size of set of large itemsets L({k}): {len(lev)}\n")
18
5        if a.I:
18
6            print(f"Large Itemsets L({k}):")
18
7            for iset in sorted(lev, key=sorted):
18
8                print(" ".join(item_str(attrs, i) for i in sorted(iset)), le
18
9                print()
19
0
19
1    # rank: metric descending, then support descending, then generation orde
19
2    key = (lambda r: (-r[4], -r[3])) if a.T == 0 else (lambda r: (-r[5], -r[
19
3    rules.sort(key=key)
19
4    print("Best rules found:\n")
19
5    for i, (prem, cons, nx, nxy, conf, lift, lev_, conv) in enumerate(rules[
19
6        lhs = " ".join(item_str(attrs, x) for x in sorted(prem))
19
7        rhs = " ".join(item_str(attrs, x) for x in sorted(cons))
19
8        m = [f"conf:({fmt(conf, 2))}", f"lift:({fmt(lift, 2))}",
19
9            f"lev:({fmt(lev_, 2))} [{int(round(lev_ * n, 6))}]", f"conv:({f
20
0        m[a.T] = "<" + m[a.T] + ">"
20
1        print(f"{i:2d}. {lhs} {nx} ==> {rhs} {nxy} {' '.join(m)}")
20
2
20
3

```

```

20
4  if __name__ == "__main__":
20
5      main()

```

Output

```
python3 apriori.py contact-lenses.arff -M 0.2 -U 0.2 (support 0.2, confidence 0.9, one pass):
```

```

1  Minimum support: 0.2 (5 instances)
2  Minimum metric <confidence>: 0.9
3  Number of cycles performed: 1
4
5  Generated sets of large itemsets:
6
7  Size of set of large itemsets L(1): 11
8
9  Size of set of large itemsets L(2): 21
10
11 Size of set of large itemsets L(3): 6
12
13 Best rules found:
14
15  1. tear-prod-rate=reduced 12  $\implies$  contact-lenses=none 12    <conf:(1)> lift:
16  2. spectacle-prescrip=myope tear-prod-rate=reduced 6  $\implies$  contact-lenses=non
17  3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6  $\implies$  contact-len
18  4. astigmatism=no tear-prod-rate=reduced 6  $\implies$  contact-lenses=none 6    <co
19  5. astigmatism=yes tear-prod-rate=reduced 6  $\implies$  contact-lenses=none 6    <c
20  6. contact-lenses=soft 5  $\implies$  astigmatism=no 5    <conf:(1)> lift:(2) lev:(0
21  7. contact-lenses=soft 5  $\implies$  tear-prod-rate=normal 5    <conf:(1)> lift:(2)
22  8. tear-prod-rate=normal contact-lenses=soft 5  $\implies$  astigmatism=no 5    <con
23  9. astigmatism=no contact-lenses=soft 5  $\implies$  tear-prod-rate=normal 5    <con
24  10. contact-lenses=soft 5  $\implies$  astigmatism=no tear-prod-rate=normal 5    <con

```

Cross-check against WEKA. Explorer, Open file `contact-lenses.arff`, Associate, Apriori with defaults, Start prints:

```
1 Minimum support: 0.2 (5 instances)
2 Minimum metric <confidence>: 0.9
3 Number of cycles performed: 16
4
5 Generated sets of large itemsets:
6
7 Size of set of large itemsets L(1): 11
8
9 Size of set of large itemsets L(2): 21
10
11 Size of set of large itemsets L(3): 6
12
13 Best rules found:
14
15 1. tear-prod-rate=reduced 12  $\implies$  contact-lenses=none 12    <conf:(1)> lift:
16 2. spectacle-prescrip=myope tear-prod-rate=reduced 6  $\implies$  contact-lenses=non
17 3. spectacle-prescrip=hypermetrope tear-prod-rate=reduced 6  $\implies$  contact-len
18 4. astigmatism=no tear-prod-rate=reduced 6  $\implies$  contact-lenses=none 6    <co
19 5. astigmatism=yes tear-prod-rate=reduced 6  $\implies$  contact-lenses=none 6    <c
20 6. contact-lenses=soft 5  $\implies$  astigmatism=no 5    <conf:(1)> lift:(2) lev:(0
21 7. contact-lenses=soft 5  $\implies$  tear-prod-rate=normal 5    <conf:(1)> lift:(2)
22 8. tear-prod-rate=normal contact-lenses=soft 5  $\implies$  astigmatism=no 5    <con
23 9. astigmatism=no contact-lenses=soft 5  $\implies$  tear-prod-rate=normal 5    <con
24 10. contact-lenses=soft 5  $\implies$  astigmatism=no tear-prod-rate=normal 5    <con
```

Check	Python, one pass at 0.2	WEKA defaults
Minimum support	0.2 (5 instances)	0.2 (5 instances), reached after 16 cycles from 1.0
L(1), L(2), L(3)	11, 21, 6	11, 21, 6
Rules with confidence at least 0.9	10	10
Rule 1	tear-prod-rate=reduced 12 $\Rightarrow$ contact-lenses=none 12	same
Rules 2 to 5	the four 2-item premises containing reduced, support 6	same four; WEKA may print them in a different order among themselves
Rules 6 to 10	the five soft rules with support 5	same

The only difference WEKA can show is the order of rules that tie on both confidence and support; the sets are identical.

Explanation

The program is the textbook algorithm. `large_itemsets` is the level-wise pass: `Counter` gives L1, `itertools.combinations` over the previous level joins pairs, the `all(... in prev ...)` test is the pruning step that throws away a candidate whose subset is not large, and one scan of the transactions counts what is left. Pruning matters even here: from 21 large pairs there are 210 possible joins, but only candidates whose three pairs are all large get counted, and only 6 survive. `rules_from` walks every large itemset and every consequent, and the confidence test $\text{conf}(X \Rightarrow Y) = \frac{|X \cup Y|}{|X|}$ is one division. The support-lowering loop in `main` reproduces WEKA's search: start at `upperBoundMinSupport` minus `delta`, mine, and lower the support until `numRules` rules pass or `lowerBoundMinSupport` is hit; 16 steps of 0.05 from 0.95 land on 0.2, which is why WEKA prints 16 cycles.

Viva Questions

✗ Do not copy. Read for understanding and the viva

Q: Why does WEKA report Minimum support 0.2 when the lower bound was 0.1? **A:** It starts at 1.0 and lowers by delta 0.05 each cycle; at 0.2 it already had 10 rules, so it stopped.

Q: What is the Apriori property? **A:** Every subset of a frequent itemset is frequent, so a candidate with an infrequent subset can be dropped without counting it.

Q: Why must numeric attributes be discretised before Apriori? **A:** An item is an attribute-value pair; a continuous value would be its own item with support 1 of N.

Q: What does the number after the arrow mean in `tear-prod-rate=reduced 12  $\Rightarrow$  contact-lenses=none 12`? **A:** Twelve rows contain both sides; the 12 before the arrow is the premise count; confidence is 12 divided by 12.

Q: What does lift 4 on rule 10 tell you? **A:** The consequent (astigmatism=no and tear-prod-rate=normal) occurs in 6 of 24 rows, 25 percent; among soft-lens rows it occurs 100 percent, four times as often.

Q: How does the number of bins change the rules? **A:** More bins mean rarer, more precise items; fewer rules pass the support threshold and their intervals are narrower.

Q: Why does the equal-width Discretize label read '(-inf-0.9]' and not '[0.1-0.9]'? **A:** WEKA makes the first and last bins open so that unseen values outside the training range still fall into a bin.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Running Apriori on iris or glass without Discretize; WEKA greys out Start because Apriori cannot handle numeric attributes.
- Setting `lowerBoundMinSupport` above `upperBoundMinSupport`, which gives no cycles and no rules.
- Reading `conf:(1)` as proof of a strong pattern when the support is 2 rows out of 14.
- Confusing the item count in the rule (rows matched) with the support fraction; divide by the number of instances.

- Forgetting that Discretize on `first-last` leaves the nominal class alone, then reporting that the class was not binned as an error.

Formula Sheet

✗ Do not copy. Read for understanding and the viva

Association rules

For a rule $X \Rightarrow Y$ over N transactions:

$$\text{support}(X \Rightarrow Y) = \frac{|X \cup Y|}{N}, \quad \text{confidence}(X \Rightarrow Y) = \frac{|X \cup Y|}{|X|}, \quad \text{lift}(X \Rightarrow Y) = \frac{\text{confidence}(X \Rightarrow Y)}{\text{support}(Y)}$$

WEKA's Apriori starts at the upper bound of minimum support and lowers it by the delta each pass until the requested number of rules is found or the lower bound is reached.

Entropy, information gain and Gini

For a set S with class proportions $p_1, \dots, p_c$:

$$H(S) = - \sum_{i=1}^c p_i \log_2 p_i, \quad \text{Gain}(S, A) = H(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} H(S_v), \quad \text{Gini}(S) = 1 - \sum_{i=1}^c p_i^2$$

ID3 splits on the attribute with the highest gain; J48 (C4.5) uses the gain ratio

$\text{Gain}(S, A) / \text{SplitInfo}(S, A)$ where $\text{SplitInfo}(S, A) = - \sum_v \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$.

Classifier evaluation

From the confusion matrix with true positives TP , false positives FP , false negatives FN , true negatives TN :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Kappa compares observed agreement p_o (accuracy) with the agreement expected by chance p_e :

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \quad p_e = \sum_i \frac{(\text{row}_i \text{ total}) (\text{column}_i \text{ total})}{N^2}$$

The ROC curve plots true positive rate $TP / (TP + FN)$ against false positive rate $FP / (FP + TN)$; the area under it (AUC) is 0.5 for guessing and 1.0 for a perfect classifier.

Naive Bayes and k-nearest neighbour

$$P(C | x_1, \dots, x_n) \propto P(C) \prod_{i=1}^n P(x_i | C)$$

k-NN assigns the majority class among the k nearest training records under Euclidean distance

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

after normalising each attribute to $[0, 1]$ with $x' = (x - x_{\min}) / (x_{\max} - x_{\min})$.

Linear regression

$$\hat{y} = \beta_0 + \beta_1 x, \quad \beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}, \quad \beta_0 = \bar{y} - \beta_1 \bar{x}$$

WEKA reports the correlation coefficient, mean absolute error and root mean squared error $\sqrt{\frac{1}{N} \sum (y_i - \hat{y}_i)^2}$.

Clustering

k-means minimises the within-cluster sum of squared errors over clusters $C_1, \dots, C_k$ with centroids μ_j :

$$SSE = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \mu_j\|^2, \quad \mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$$

Hierarchical (agglomerative) clustering merges the two closest clusters each step; linkage defines closeness: single $\min d(a, b)$, complete $\max d(a, b)$, average $\frac{1}{|A||B|} \sum d(a, b)$.

DBSCAN calls a point a core point when at least minPts points lie within radius ϵ ; clusters grow from core points, and points reachable from none are noise.

Session Summary

Write in lab record

- Question 7: unsupervised filters catalogued with menu paths; Apriori run on weather.nominal (defaults and 0.3 with 0.7), on iris with 3 and 10 bins and on glass; rules interpreted and the effect of bin count recorded

- Question 8: `apriori.py` written and run on `contact-lenses.arff` at support 0.2 and confidence 0.9; L1, L2, L3 = 11, 21, 6 and all 10 rules match WEKA's default output

 [Edit this page](#)

[< Previous](#)
Session 2

Session 4 [Next >](#)